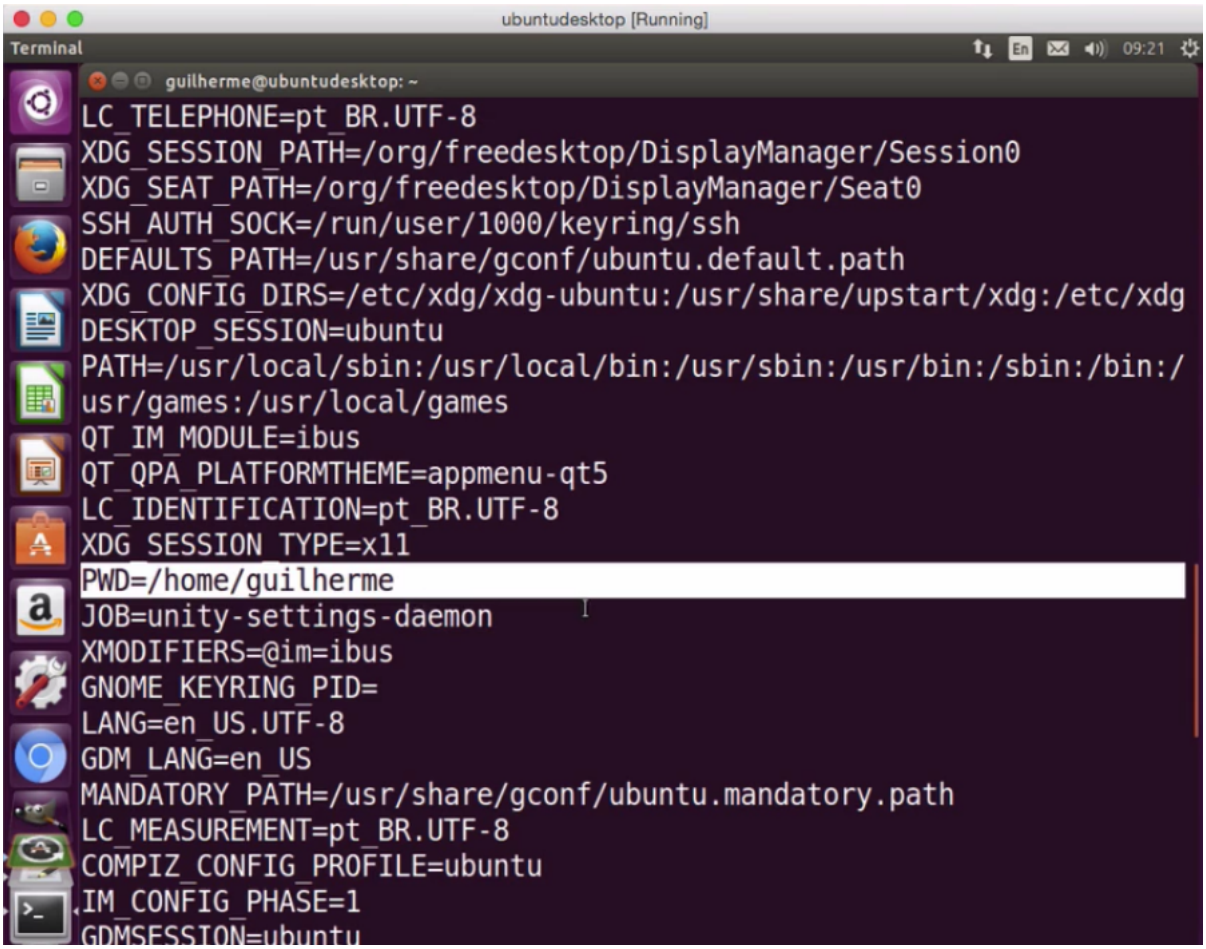
03

As variáveis de ambiente PWD, PS1 e PS2

Tentando enganar o pwd e ps1 e ps2

Agora que já vimos como criar e remover diversos tipos de variáveis tanto de *shell* quanto de *ambiente* das mais diversas maneiras, Vamos dar uma olhada em outras variáveis que existem.

Para isso vamos começar com uma variável bem bacana que é a `pwd`. Vamos buscar na lista de variáveis de ambiente do `env` a `pwd`:



Encontraremos `PWD=/home/guilherme`, o `/home/guilherme` indica nosso diretório atual. Vamos dar um `clear` nessa tela.

Bom, veremos a `pwd`, mas, ela não soa familiar? Lembra-se que já passamos brevemente por esse comando antes?

Vamos digitar `pwd` e ver o que acontece:

```
> pwd
/home/guilherme
```

Ele nos mostra `/home/guilherme` que é nosso diretório atual. E se dermos um `echo $PWD`? Teremos:

```
> pwd
/home/guilherme
```

```
> echo $PWD
/home/guilherme
```

Toda vez que mudamos de diretório essa variável do ambiente, a `PWD` é setada de acordo com o diretório atual. Por exemplo, se chamarmos o `ls` e logo em seguida utilizarmos o `cd desktop`, onde `cd` (*change directory*) serve para mudar o diretório, estaremos mudando para o `desktop`.

Feito isso, mudaremos de diretório. Agora, se formos buscar o `echo $PWD` teremos como resposta: `/home/guilherme/Desktop`. Vamos ver o que teremos:

```
> pwd
/home/guilherme
> echo $PWD
/home/guilherme
> ls
desktop     examples.desktop    Music           Templates
Documents   mostra_idade        pictures        Videos
Downloads   mostra_idade        Public
> cd Desktop
~/Desktop$ echo $PWD
/home/guilherme/Desktop
~/Desktop$ pwd
/home/guilherme/Desktop
```

Então, o `pwd` é o nome do diretório atual? O que acontece se mudarmos a variável? Se usarmos o `cd ..` estaremos retornando ao nosso diretório primeiro, o `/home/guilherme`. E se pedirmos para o `pwd` qual é nosso diretório ele nos responderá:

```
> cd ..
> pwd
/home/guilherme
```

Ao em vez de falar `cd Desktop` e mudar o diretório através do `cd`, *change directory* podemos simplesmente usar o `PWD=/home/guilherme/Desktop`. Teremos a seguinte resposta:

```
> PWD=/home/guilherme/Desktop
~/Desktop$
```

Aparentemente mudamos do diretório atual para o `Desktop`. Vamos observar os arquivos do nosso diretório através do `ls`:

```
~/Desktop$ ls
Desktop     examples.desktop    Music   Templates
Documents   mostra_idade        Pictures    Videos
Downloads   mostra_idade        Public
```

Com isso percebemos que não entramos no diretório `Desktop` e para comprovar isso basta dar mais um `pwd`:

```
> pwd
/home/guilherme
```

Ainda estamos no diretório `/home/guilherme`. Alterar a variável `pwd` é como roubar, isto é, se alteramos a variável `pwd` não alteramos o diretório atual. Para alterar o diretório atual usamos o `cd` e um espaço, e o diretório onde queremos ir, por exemplo:

```
~/Desktop$ cd /home/guilherme
pwd
/home/guilherme
```

Podemos verificar que agora mudamos, efetivamente, de diretório, pois, o pwd nos diz que estamos no `/home/guilherme`. Vamos ir para o diretório `Desktop` outra vez? E vamos aproveitar para verificar se estamos no diretório que escolhemos através do `echo`.

```
> cd Desktop
~/Desktop$ pwd
/home/guilherme/Desktop
~/Desktop$ echo PWD
/home/guilherme/Desktop
```

Repare, a variável `pwd` serve para nós, para nossos scripts e programas, compreendermos em qual diretório eles estão. Mas, não serve para alterar o diretório atual. Para realizar essa alteração nós utilizamos o `cd`.

As variáveis de ambiente que já vêm para nós são usadas de uma maneira muito específica. Temos que conhecer como elas ***podem*** e como ***não podem*** ser utilizadas. O `pwd` é uma dessas variáveis que serve para sabermos, nos nossos scripts, qual o diretório atual, onde estamos rodando algo. Utilizamos o `$PWD` para nos indicar isso.

O `$PWD` é bem útil e acaba sendo utilizado para compreender qual é o diretório atual. Agora, se quero mudar o diretório atual, então, devemos usar o `cd`.

Mas, vamos voltar um pouco, quando utilizamos o `PWD=/home` enganamos o terminal, o console, pois "fingimos" que estavamos no diretório home, inclusive, quando damos o `echo PWD` ele nos responde:

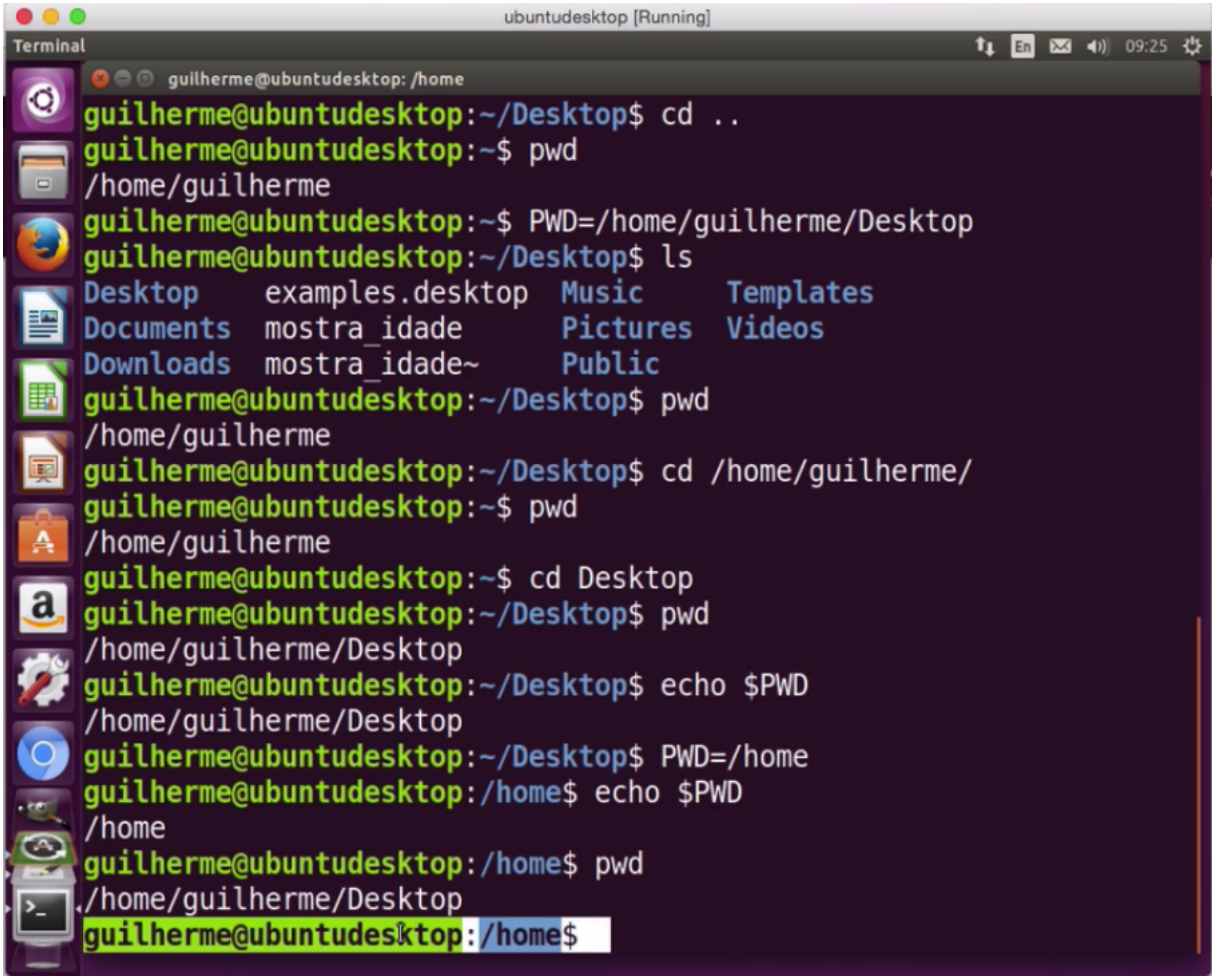
```
~/Desktop$ PWD=/home
/home$ echo PWD
/home
```

Ele nos responde que estamos, de fato em home, o que não é verdade, pois estamos no diretório `/home/guilherme/Desktop`:

```
~/Desktop$ PWD=/home
/home$ echo PWD
/home
pwd
/home/guilherme/Desktop
```

Repare que até mesmo o `console` nos mostra no *Ubuntu* que estamos no `/home`.

Temos que ter um cuidado. Quem configura o que está sendo dito?



No nosso *Ubuntu Desktop* temos:

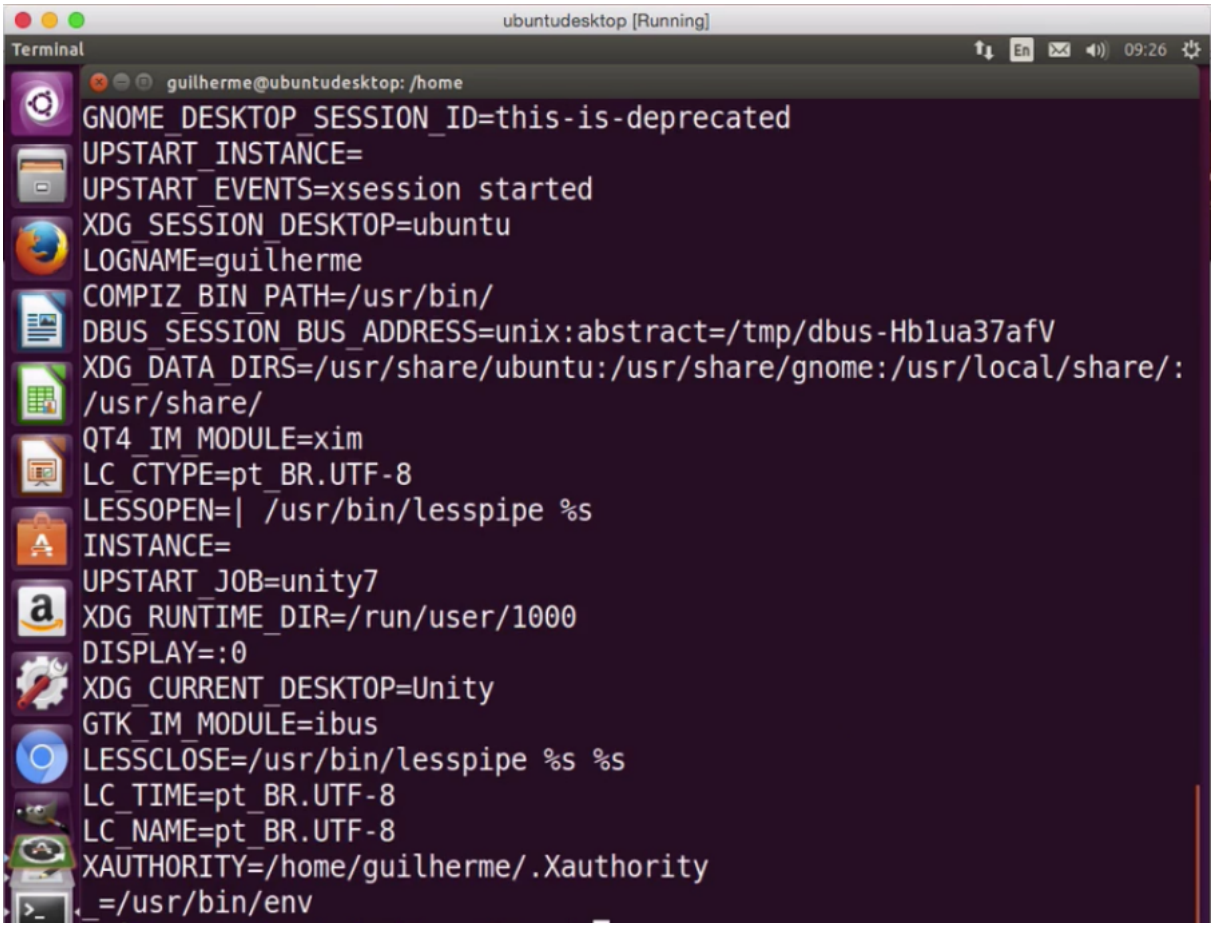
```
nome do usuário, @, nome da máquina local, :, o diretório que estamos, $, espaço, e o cursor para digitar o comando.
```

Essas são as informações que permitem que possamos escrever algo, é o *promt* do comando. O *promt* do comando é: nome do usuário, @, nome da máquina local, :, o diretório que estamos, $, espaço, e o cursor para digitar o comando.

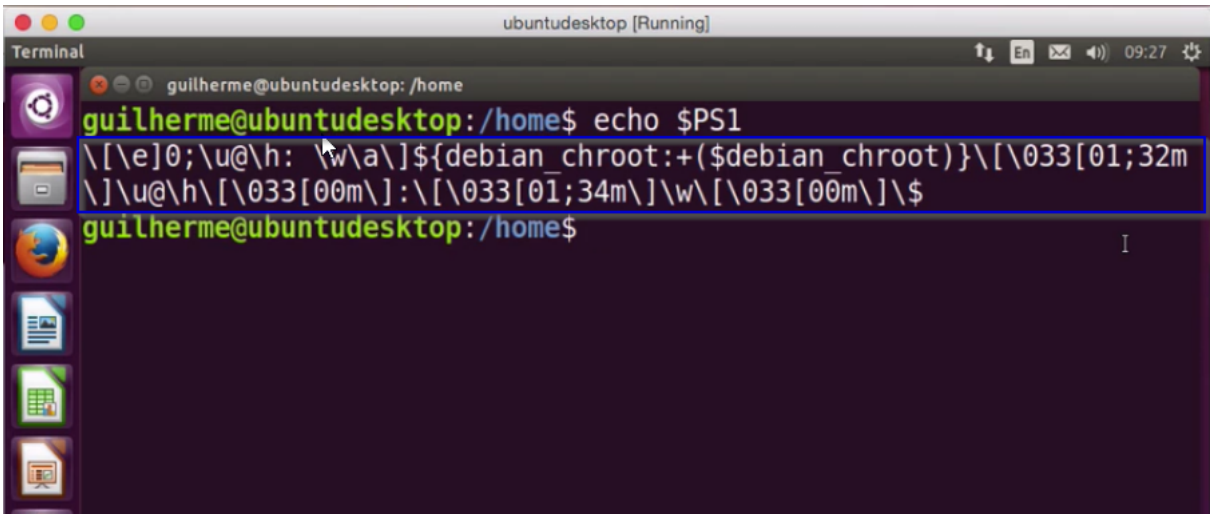
Em cada máquina podemos encontrar um padrão distinto, por exemplo, no *Fedora* será utilizado, provavelmente, outro padrão. Quem configura isso? Quem está sendo enganado pela variável `pwd` ?

Vamos observar!

Vamos limpar atela utilizando um `clear` e vamos usar uma variável chamada `ps1` . Digitando ela teremos o seguinte:



O `ps1` é o `prompt 1` , existem outros *prompts*, mas primeiro observaremos o *prompt 1*. Se buscarmos ele na lista da imagem acima não encontraremos, vamos dar um `clear` e um `echo $PS1` . Teremos o seguinte:



Parece bastante complicado, mas o que queremos entender é o comando que ele utiliza para configurar tudo isso. Não precisamos decorar nada disso que ele traz, na verdade, o `PS1` não aparece como um conteúdo explícito da prova. Mas, é uma variável extremamente importante no dia a dia pois ela diz onde estamos e o que está acontecendo. Existem diversas maneiras de configurar essa variável, uma maneira mais simples é setar uma variável, para isso, `PS1` é `=` a "o que quisermos".

Por exemplo, se quiser falar que essa é a máquina do Guilherme, para isso digitaremos `guilherme:` . Ficaremos com `PS1=guilherme:` . Teremos:

```
> PS1=guilherme:
guilherme:
```

Vamos observar como está o *prompt* e observar como está tudo funcionando. Vamos dar um `ls` para listar os arquivos, um `pwd` para saber o diretório, vamos entrar no diretório, vamos pedir `pwd` de novo e vamos dar um `ls`. Repare:

```
> guilherme:ls
> guilherme:pwd
/home/guilherme/Desktop
guilherme: cd/home/guilherme/
guilherme: pwd
/home/guilherme
guilherme:ls
Desktop         examples.desktop        Music           Templates
Documents       mostra_idade            Pictures        Videos
Dowloads        mostra_idade            Public
```

Tudo funcionando ok! Só deixou de ser aquela coisa estranha que o *Ubuntu* tinha colocado para nós, cheia de informações e colorida para simplesmente `guilherme:` o que é bem pobre para `PS1`.

Então, `PS1` poderia ser outras coisas além de `guilherme:`, poderíamos utilizar referências mais simples como, `PS1= $ `. Ficaremos com:

```
> guilherme:PS1=`$ `
$ ls
Desktop         examples.desktop        Music           Templates
Documents       mostra_idade            Pictures        Videos
Dowloads        mostra_idade            Public
$ pwd
/home/guilherme
```

Agora, não temos mais as várias informações que tínhamos em nossa tela. Esse modelo do `$` é adotado, normalmente, para os cursos que são gravados. Assim, o aluno não se distrai.

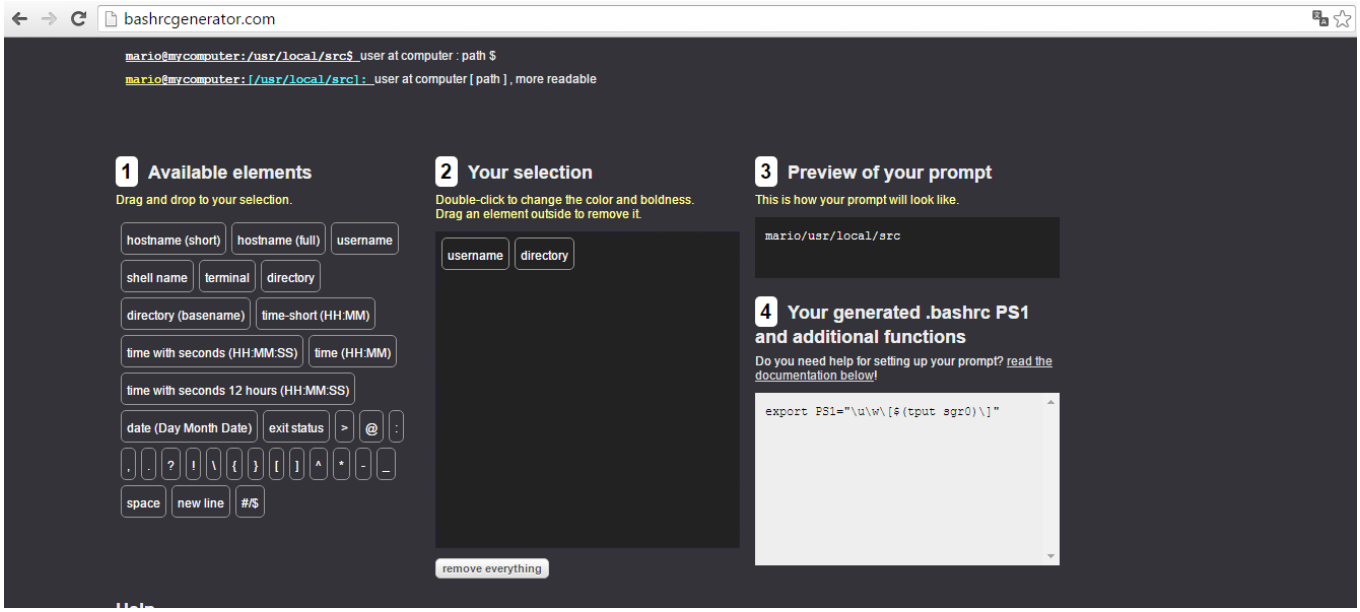
É óbvio que nesse curso de *Linux*, onde estamos falando de *prompts* e diversas outras informações, não utilizaremos esse `PS1`. Uma vez que as outras informações, o diretório atual, o usuário atual, a máquina própria são informações ricas e importantes para nós. Por isso, utilizaremos outros `PS1`.

O `PS1` é também uma variável de ambiente usada para controlar o que o terminal, o console, nos mostra cada vez que nos indica um *prompt* de informações. Bom, esse é o `PS1`.

Temos como configurar esse `PS1` para diversas outras coisas? Sim!

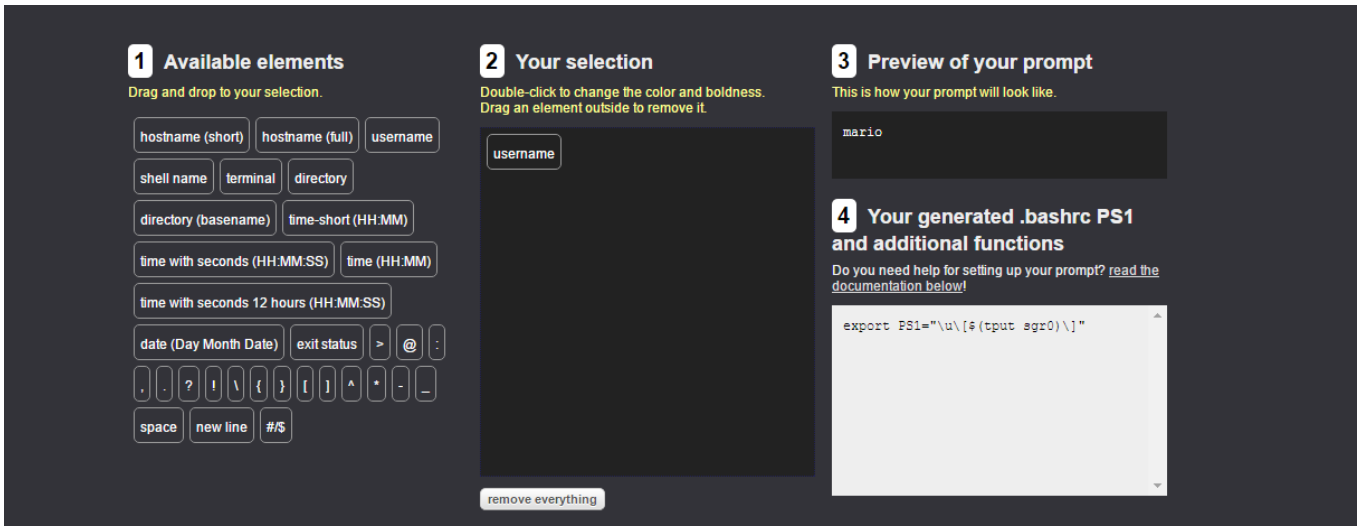
Vamos buscar no google *ps1 linux configuration* e vamos entrar na seguinte página: http://bashrcgenerator.com/ (http://bashrcgenerator.com/). Nessa página encontraremos um bom exemplo.

Repare na página:



Nessa página podemos simplesmente arrastar da coluna um, que é "Avaible elements", para a coluna dois, "Your selection". Movemos o que queremos no nosso *bash*.

Por exemplo, queremos o nome do usuário, arrastaremos o `username`. Na coluna 4 ele nos falará para realizar um `export PS1=\u\` . Além do `\u\` que é um "u" de usuário ele nos fala o seguinte `[$(tput sgr0)\]"`. Mas, vamos ignorar nesse momento essas outras coisas.



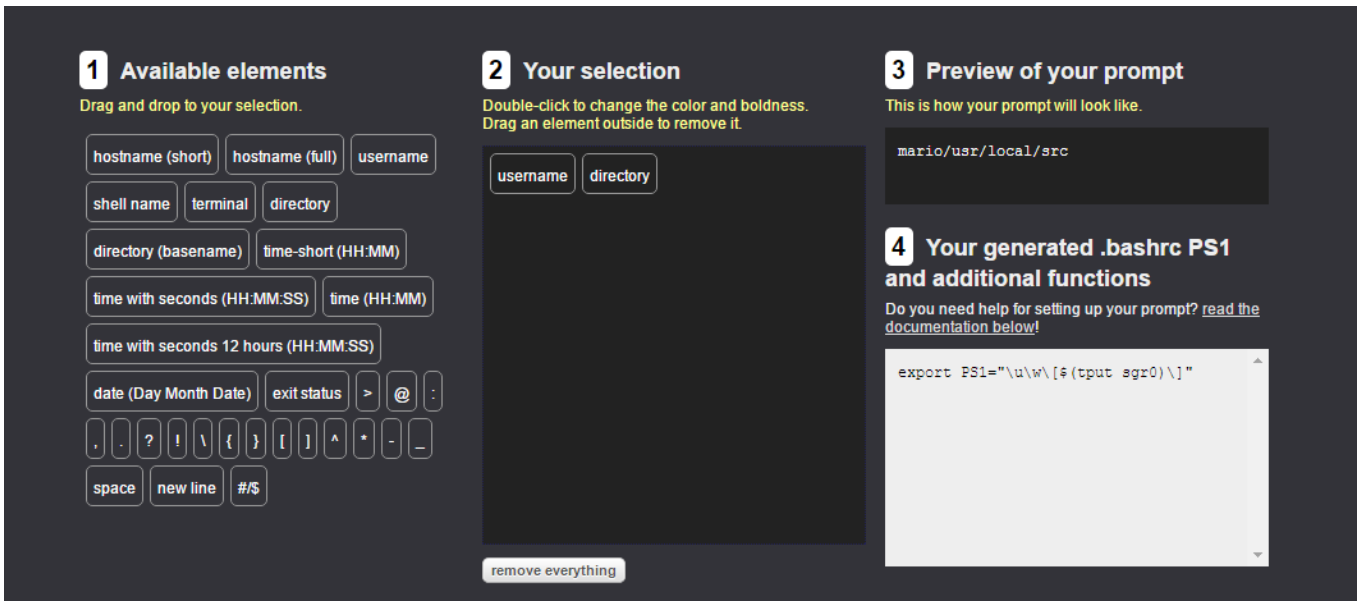
Vamos testar a `\u\` de usuário?

Voltamos para o terminal e digitamos o "u", mas, ao em vez de utilizarmos duas barras usaremos apenas uma barra `"\"` e dois pontos `":"`. Teremos `$PS1=` \u: .

```
> $ PS1=`\u: `
guilherme:
```

Ele devolve o nome do usuário, `guilherme:`. Se fosse outro usuário ele devolveria outro nome.

Vamos voltar na página e ver o que acontece se arrastarmos a palavra diretório ou como está em inglês *directory*, da coluna 1 para a coluna 2. Queremos, agora, o nosso diretório atual.



Ele nos mostrará: `export PS1=\u\w\` e outras coisas que não nos importam agora.

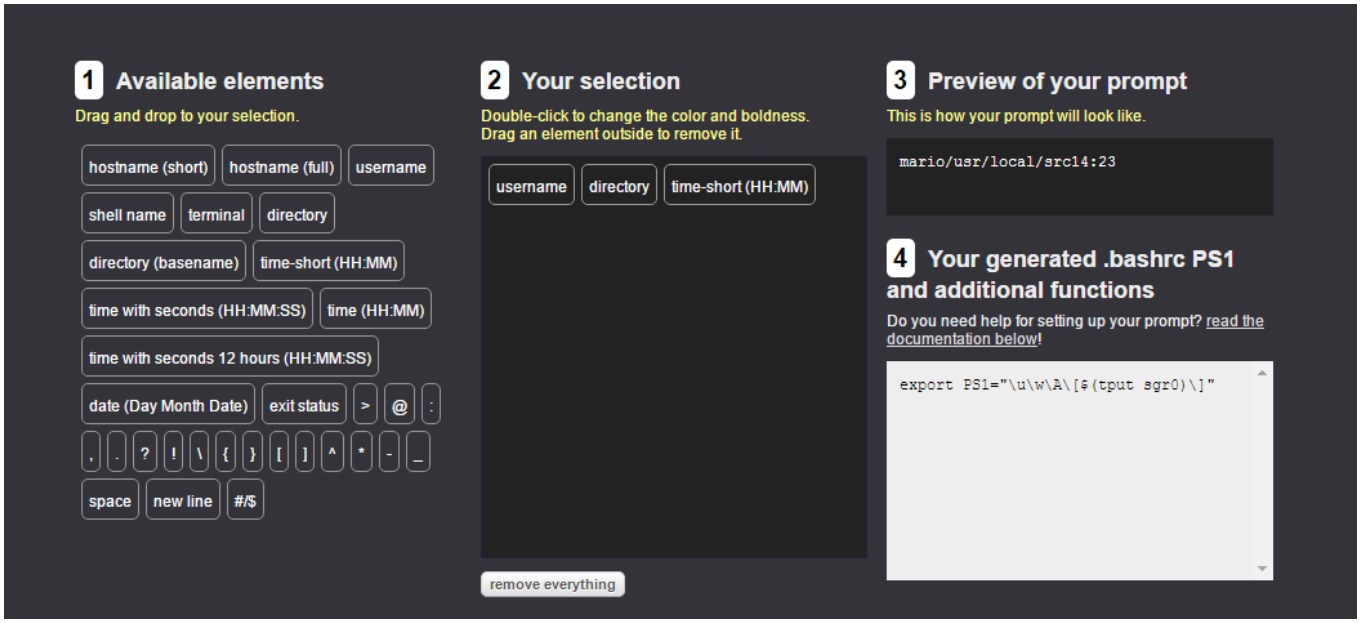
Mas, repare. Não fica um pouco estranho? Bastaria que no meio de *username* e *directory* houvesse algum carácter escrito, algum carácter específico. Nós mesmos vamos colocar isso. Digitaremos: `PS1=\u@\w: `. O "@" é o *at* e indica que estamos no diretório e usaremos o `w` para indicar o diretório. Então, ele nos responde com `guilherme@~` , isto é, com o nome do usuário, o nome do diretório e, como estamos na `home` , no diretório raiz do usuário ele mostra apenas um til, `~` , que é seu padrão. Teremos:

```
guilherme: PS1=`\u@\w: `
guilherme@
```

Se quisermos entrar no diretório `Desktop` usaremos o `cd Desktop` e ele mostrará apenas um `/Desktop` :

```
guilherme: PS1=`\u@\w: `
guilherme@~: cd Desktop
guilherme@~/Desktop:
```

Existem outras maneiras de representar o diretório, tem o diretório *basename*, tem o diretório de hora, o *time(HH:MM)*. Vamos arrastar o time(HH:MM) e ver como ele funciona!



Ele pede um `\@\` .Vamos testar o `\@\` , para isso, digitaremos novamente o `\u\` de usuário, o símbolo de *at*, `@` e acrescentaremos o `\@:` para mostrar a hora atual. Iremos digitar: `guilherme@~/Desktop: PS1=` \u@\@: `.

Teremos como resposta:

```
guilherme@~/Desktop: PS1=`\u@\@: `
guilherme@9:31 :
```

Podemos ver que ele nos passa o horário atual, por exemplo, 9:31. Podemos criar qualquer coisa que quisermos e esse site que estamos vendo vai dando as dicas. Temos diversas configurações e não é necessário que se decore tudo, você pode utilizá-las a medida que for sendo necessário, a medida que for achando que é interessante uma ou outra variável.

Essa é a configuração do `PS1` : uma variável de ambiente para a linha de comando que pergunta quais informações eu quero para poder executar o comando.

Existem outros `PS` ?

Sim, existe o `PS2` , `PS3` , `PS4` e etc. Todos eles são usados de maneiras um pouco diferente.

Vamos pegar o exemplo do `PS2` . O `PS2` é um indicador de próxima linha, ele serve para quando executamos um comando e ele é muito grande e, por isso, desejamos quebrar ele em duas linhas. Por exemplo, gostaria de inserir uma mensagem, `echo "Guilherme hoje o dia esta muito lindo e voce vai sonhar um sonho muito lindo hoje a noite"` . Perceba, essa mensagem ficou muito longa, por isso vamos quebrar ela dando um "Enter" em `voce vai sonhar um sonho muito lindo hoje a noite .`

O português ficou bastante repetitivo, mas é meramente ilustrativo. Repare que utilizaremos a mensagem dentro de aspas duplas. Isso é algo que veremos mais adiante, ou seja, que tipo de aspas podemos utilizar mais adiante.

Teremos o seguinte:

```
guilherme@9:31 :"Guilherme hoje o dia esta muito lindo e
> voce vai sonhar um sonho muito lindo hoje a noite"
Guilherme hoje o dia esta muito lindo e
voce vai sonhar um sonho muito lindo hoje a noite
```

Ele irá mostrar as duas linhas, mas repare que onde demos o primeiro "Enter" aparece um carácter o `>`. Esse é o `PS2`.

Vamos alterar o seu carácter? Para isso, digitaremos: `PS2=` nova linha`>` . `Vamos testar isso usando o` echo` e usando a seguinte frase, "Guilherme hoje em dia os dias estão diariamente muito lindos". Daremos um "Enter" antes do diariamente.

```
> PS2=`nova linha>
echo `Guilherme hoje em dia os dias estao
nova linha> diariamente muito lindos`
```

Observe, que depois de `hoje em dia os dias estão` o PS2 está acontecendo. O `PS2` padrão no *Ubuntu Desktop* é o símbolo de `>`. Aqui, nós alteramos esse símbolo para que ele fosse `nova linha>`.

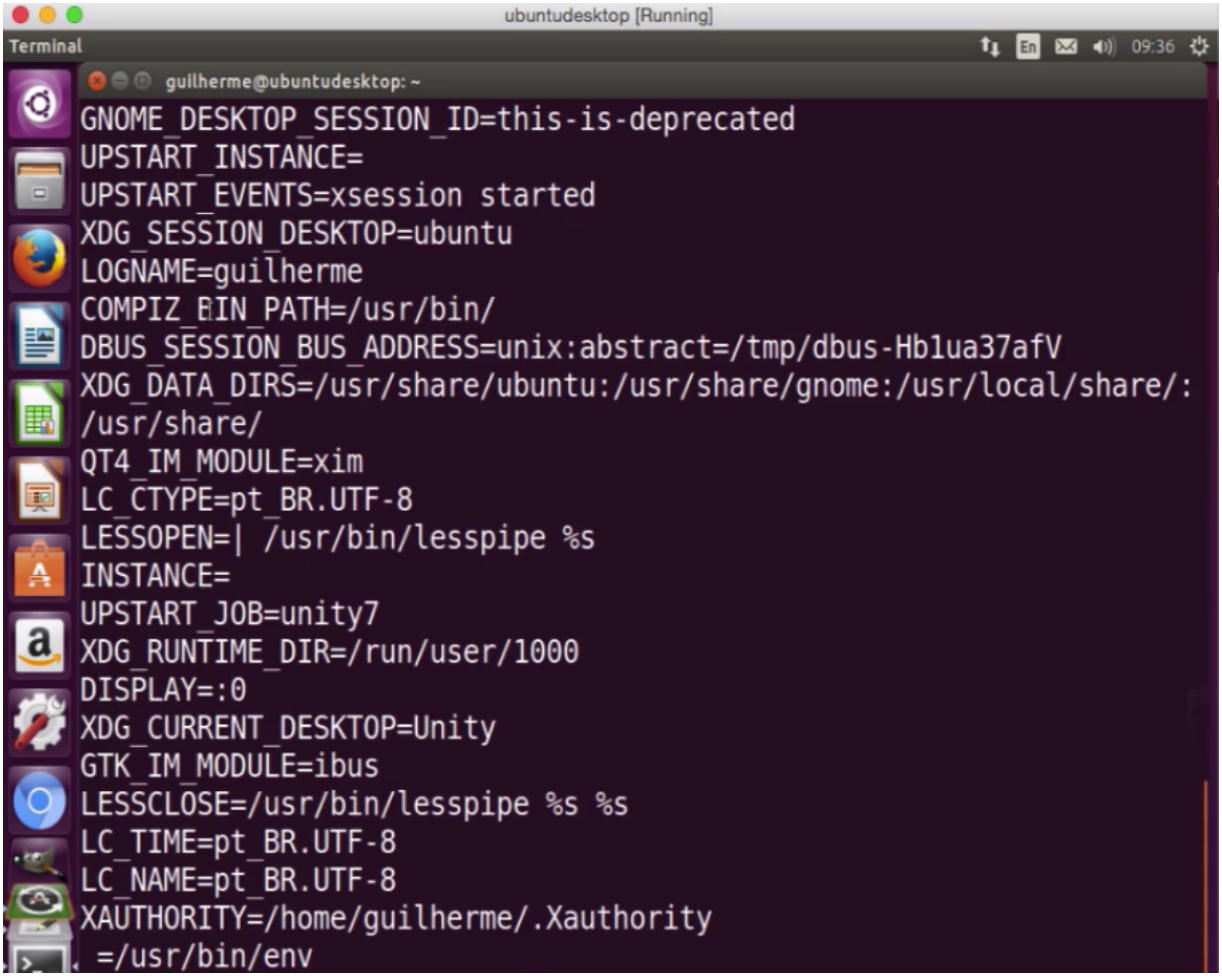
É importante observar que existem outros `PS` além desses dois que acabamos de analisar. Mas, não é necessário que nos debrucemos sobre eles com profundidade. Na verdade, a prova não cobra esse conteúdo, não cita explicitamente os `PS`. É recomendado que saibamos o que é o `PS1` pois ele é o `prompt` da linha de comando, o `\u`, que é o usuário e o `\w`, que é o diretório atual. Esses são os que mais utilizamos no dia a dia.

Logo mais falaremos sobre mais variáveis de ambiente.

Mais variáveis

Vimos algumas variáveis de ambiente que aparecem quando usamos o comando o `env`. As utilizadas no *Linux* e no *shell* em geral.

Vamos abrir um terminal novo e dar um `env` para ver algumas das outras variáveis. Teremos o seguinte:



Existem algumas outras variáveis que são muito importantes no dia a dia. Uma delas é o `HOME` que tinha sido citado anteriormente. O `HOME` indica qual o diretório raiz do usuário, os arquivos que o usuário for criando, onde eles devem ficar por padrão? A raiz deste usuário que estamos utilizando nesse presente momento, é no `/home/guilherme`. Todo *Linux* tem como padrão `/home` e o nome do usuário.

Isso pode ser configurado?

Sim, pode ser configurado e pode ser trocado.

Como descobrimos qual é?

Usamos a variável `$HOME`. Vamos fazer isso, escreveremos `echo $HOME` e observamos o que ele contesta:

```
> echo $HOME
/home/guilherme
```

Se usamos algum *script* e precisamos escrever algum arquivo no diretório que é do usuário, sabemos que a `home` dele, é dele, pertence a ele. É a partir da `HOME` onde devem ficar os arquivos pertencentes a este usuário.

A variável `$HOME` é extremamente importante no nosso dia a dia. O padrão é importantíssimo e devemos memorizá-lo. O padrão no *Linux* é `/home/nome do usuário`, exceto, para o super usuário principal da máquina que é o usuário raiz, isto é, o usuário que cria o universo da máquina. Esse usuário se chama *root* e existe um diretório especial para ele, que não é `/home/root`. O diretório do usuário raiz é, simplesmente, `/root`.

Vamos dar um `ls /root` e observar:

```
> ls /root
ls: cannot open directory/root: Permission denied
```

Se dermos um `ls` nem poderemos ver o que tem no `root` por uma questão de permissão. Falaremos sobre o tópico permissão de usuários mais adiante, quando formos abordar as questões sobre permissão de usuários.

Nesse instante é importante sabermos que o padrão do diretório raiz é `/home/nome_do_usuario`. ***Exceto*** para o `root`, cujo diretório é `/root`. Esses são os dois padrões que devemos conhecer para diretório de acordo com o usuário.

Que outras variáveis aparecem e são interessantes aqui?

Tem mais três variáveis que são interessantes. A primeira delas é o `logname`, que é o nome do usuário. A segunda é o `HOME` que nos informa qual é o diretório padrão do usuário no qual podemos trabalhar, salvar os arquivos e etc. A terceira variável é o `$UID` do usuário no sistema. Se digitarmos `echo $UID` teremos o seguinte:

```
> echo $UID
1000
```

Podemos observar que nosso usuário tem o `UID` número `1000`. O `UID` significa `user id`. A variável `UID` é o número do usuário que você estiver utilizando, que no nosso caso é "1000".

Podemos mudar essa variável? Por exemplo, alterar para "1005"?

Não podemos. Algumas variáveis não podemos mudar. O `UID` identifica nosso usuário no sistema operacional e não podemos alterar isso. Se mudássemos é quase como se estivéssemos enganando nosso sistema operacional, estaríamos dizendo que somos outro usuário e por questões de permissão, não podemos alterar essa variável.

Por isso, essa variável é especial, ela é uma variável de ambiente, mas ele é somente para leitura, portanto, não pode ser alterada. Essa é a variável `UID`, a `user id`. É ela quem identifica o código, o número do usuário nessa máquina. Não é o `id`, de nome de usuário `Guilherme` e de *login* e senha. Aquele *login* Guilherme apareceu onde? Apareceu no `LOGNAME`. No `LOGNAME` apareceu a palavra "Guilherme". Para saber quem é o usuário que está executando o programa usamos o `LOGNAME` e saberemos o nome do usuário. E se quisermos usar um `id` dele que esteja de acordo com a nossa máquina, isto é, a sequência de números sendo gerada, usaremos o `UID` que é uma variável que não podemos alterar. Então, além do `UID`, do `HOME` e do `LOGNAME`, temos mais um que gostaríamos de mostrar, que é o editor padrão.

Em algumas situações, podemos utilizar algum programa, por exemplo, o *git*, o *subversion*, que são programas que utilizam um editor para editar algum conteúdo no nosso sistema. Então, esses programas vão ter que saber qual é o editor que preferimos usar.

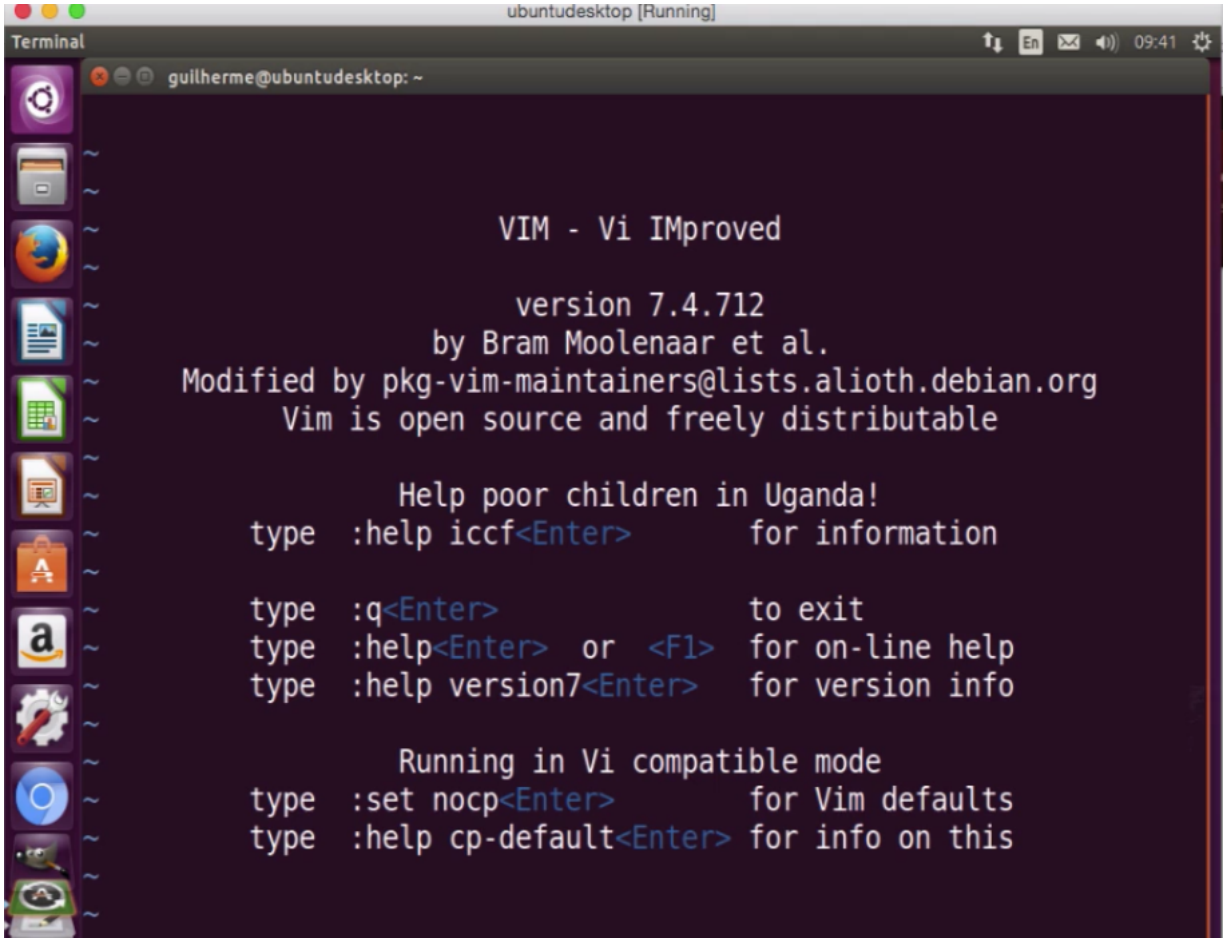
Qual é o editor que preferimos usar?

A variável que controla isso é a `$EDITOR`, de editor. Vamos digitar `echo $EDITOR` e ver o que acontece:

```
> echo $EDITOR
```

podemos observar que ela está vazia. O `$EDITOR` está vazia por padrão. Essa variável é quem define qual o editor que vamos utilizar por padrão.

Então, como podemos usar ela? Podemos digitar `EDITOR=/usr/bin/vi` e dessa maneira estamos dizendo que o nosso `$EDITOR` é o `/usr/bin/vi`. O `vi` é um editor, então, se digitamos `/usr/bin/vi` e damos um "Enter" entramos no editor do `vi`:



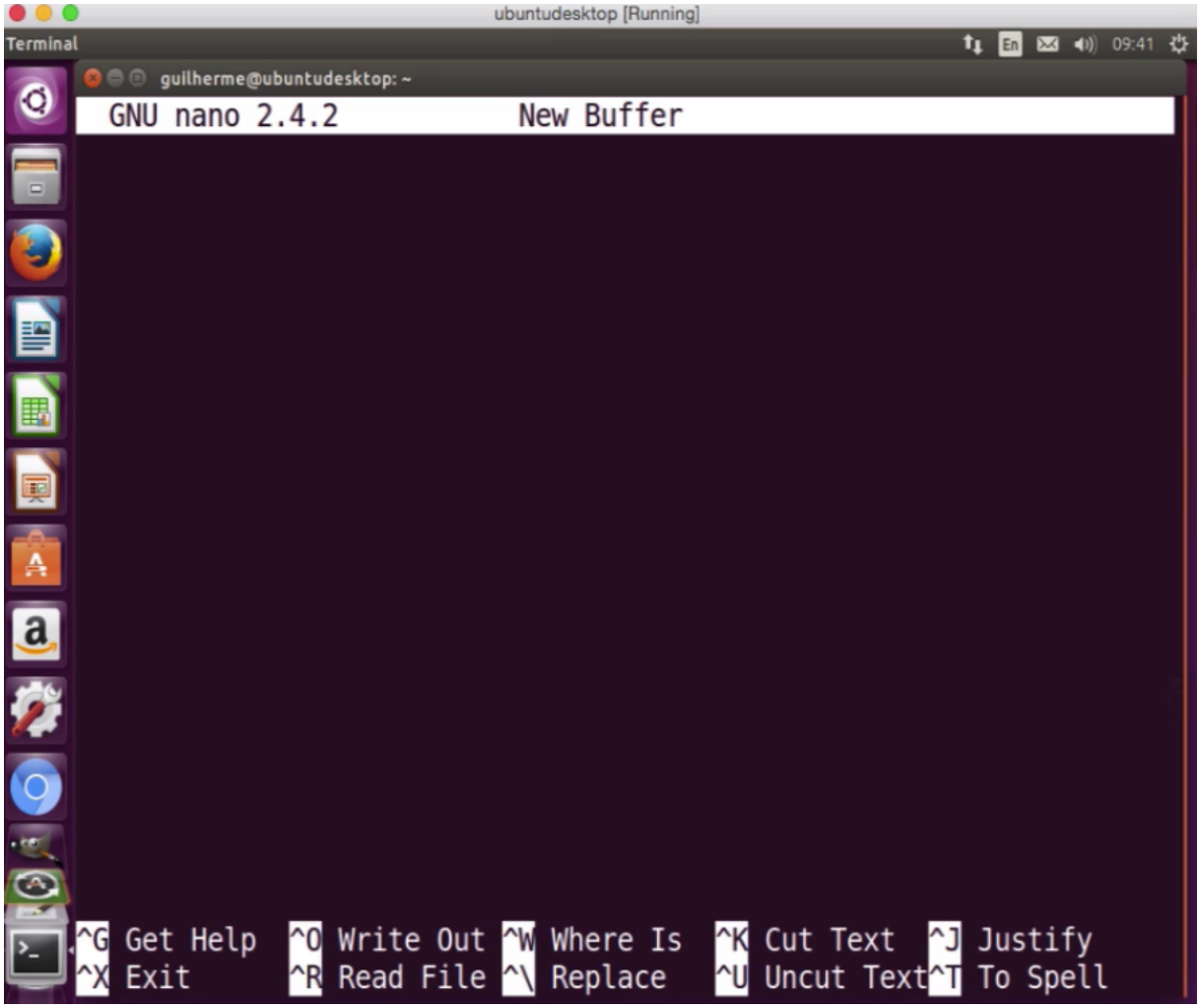
Para sair do editor do `vi`, basta digitar `:q`, seguido de um "Enter" e, pronto, sairemos do editor.

E se quiséssemos usar um outro editor, o editor do `nano`, como fazemos para entrar no editor do `nano`?

Digitamos `EDITOR=/bin/nano` e para entrar nele escrevemos `/bin/nano` e entramos no editor. Vejamos:

```
> EDITOR=/bin/nano
/bin/nano
```

O editor do `nano` é o seguinte:



Para sair desse editor basta digitarmos "Ctrl X".

Repare que o `vi` e o `nano` costumam vir instalados com as distribuições *Linux* e podemos escolher o editor que vamos utilizar em um ou outro programa. E como fazemos para escolher isso? Utilizamos a variável `EDITOR`.

Repare que são apenas alguns programas que utilizam essa variável, como citamos, o `git`, o `subversion` que é `svn` na linha de comando. Vamos ver o que aparece quando digitamos `git`, `subversion` e `svn`:

```
> git
The program 'git' is currently not installed. You can install it by typing:
sudo apt-get install git
> subversion
subversion: command not found
> svn
The program `svn` is currently not installed. You can install it by typing
sudo apt-get install version
```

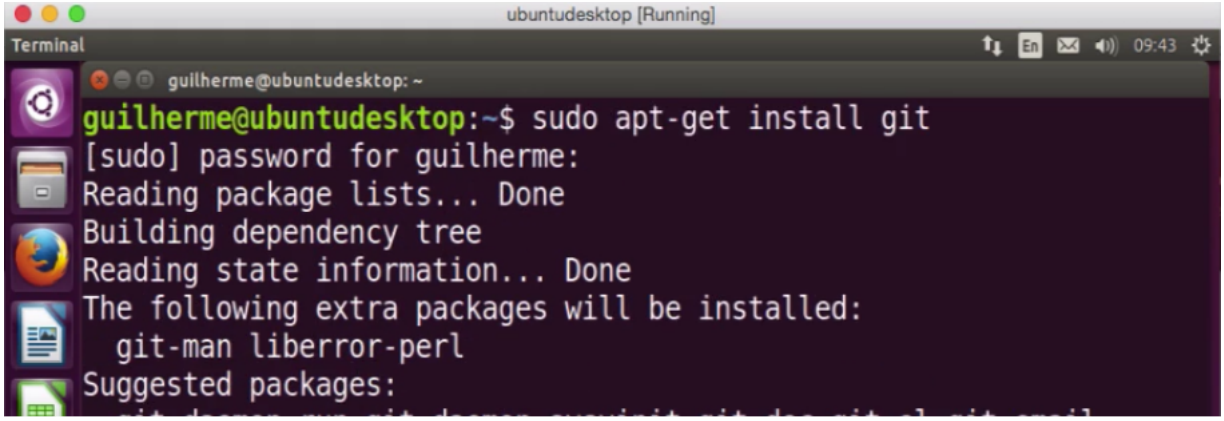
Repare que eles dois não estão instalados no nosso *Ubuntu Desktop* por padrão, teríamos que instalar eles para poder utilizar isso.

Como gostaríamos de mostrar a variável sendo utilizada, vamos instalar o `git`. Não se preocupe em saber como funciona o `git`, ou o que ele faz no dia a dia. Isso não é cobrado na prova, esse servidor de repositório remoto não é cobrado.

Mas como queremos mostrar isso vamos instalar o `git` digitando `sudo apt-get install git` e damos um "Enter". Ele vai pedir nossa senha e digitaremos ela. Teremos o seguinte:

```
> sudo apt-get install git
```

Vejamos como fica a tela após finalizar a instalação:



O `git` foi instalado. Podemos limpar a tela utilizando o `clear`. Vamos criar um diretório onde possamos trabalhar com o `git`. Para criar um diretório digitamos `mkdir` e o nome desse diretório se chamará o nome de um projeto próprio nosso. Como nosso projeto é uma loja, chamaremos de "loja" e entraremos nele usando o `cd loja`. Teremos o seguinte:

```
> mkdir loja
> cd loja
~/loja$
```

Aqui dentro desse diretório, vamos pedir ao `git` para inicializar o programa `git`. Para iniciar o `git` digitamos `git init`. Ficaremos com:

```
~/loja$ git init
Initialized empty Git repository in /home/guilherme/loja/.git
```

Dentro desse diretório vamos salvar um arquivo. Vamos abrir nosso editor e criar um novo arquivo, nele escreveremos `Bem vindo a nossa loja` e salvaremos esse arquivo como se ele fosse um arquivo `.html`. O arquivo ficará com o seguinte conteúdo:

```
<html>
Bem vindo  nossa loja
</html>
```

Algo bem simples, apenas mencionando que este arquivo é um `html` . Vamos entrar no "loja>" e salvar com o nome de "bemvindo.html". Se dermos um `ls` encontraremos ele:

```
> ls
bemvindo.html
```

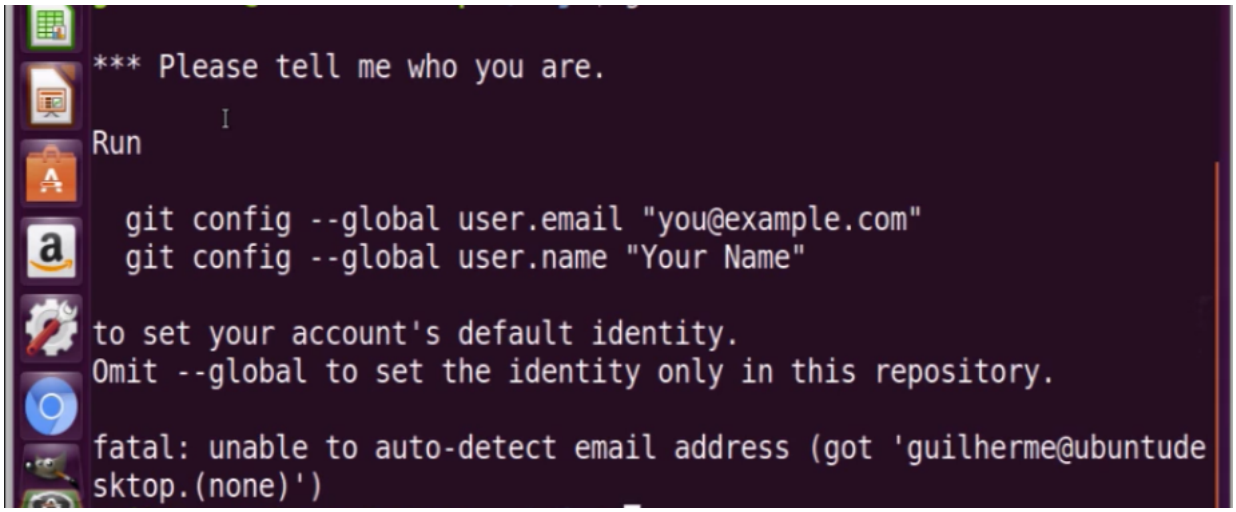
Como fazemos, agora, para adicionar esse arquivo do `git` ?

Digitaremos `git add bemvindo.html` .

```
~/loja$ git addd bemvindo.html
```

E como fazemos para comentar esse arquivo? Para falar que as mudanças feitas nesse diretório, nesse repositório do `git` devem ser registradas?

Para registrar digitamos `git commit` e ao darmos um "Enter" teremos o seguinte:



Ele está pedindo para que digamos quem nós somos, `***Please tell me who you are` . Portanto, temos que configurar no `git` quem somos nós. Fazemos isso respondendo ao que ele nos pergunta: `git config --global user.email "yu@example.com"` e `git config --global user.name "Your name"` .

No nosso caso digitaremos o seguinte e executaremos:

```
~/loja$ git config --global user.email "guilherme.silveira@alura.com.br"
~/loja$ git config --global user.name "guilherme.silveira"
```

Vamos limpar a tela e digitarmos, novamente, `git commit` . Teremos o seguinte:

```
<img src="https://s3.amazonaws.com/caelum-online-public/essentiallinux3/6_4/6_4_6nano.png">
```

Ele abriu o `nano`. Vamos sair do `nano`. Digitaremos "Ctrl X".

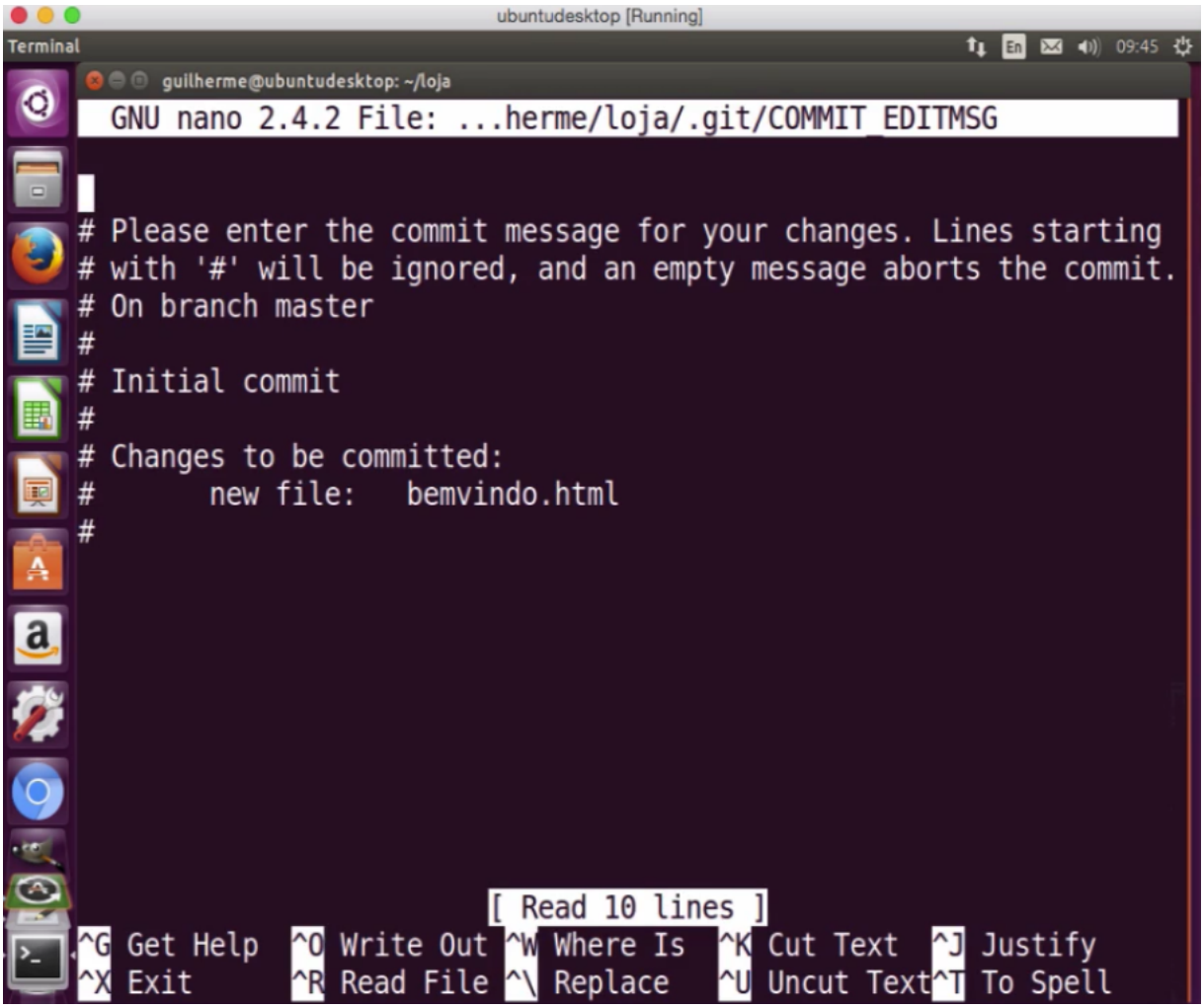
E temos no nosso terminal o seguinte:

```
~/loja$ git commit
Aborting commit due to empty commit message.
```

Como podemos observar, ele não faz nada, pois, não colocamos nenhuma mensagem. E se mudarmos o editor? E se alterarmos para `EDITOR=/usr/bin/vi`? Vamos dar um `git commit` na sequência e ver o que acontece:

```
~/loja$ EDITOR=/usr/bin/vi
~/loja$ git commit
```

Ele abre o `nano`, novamente.



Vamos sair utilizando "Ctrl X". O que será que esquecemos?

Faltou dizermos que a variável `EDITOR=/usr/bin/vi` deve poder ser acessada pelos processos filhos do processo atual. Criamos uma variável de *shell* e temos que falar que essa variável deve ser exportada. Portanto, digitamos `export EDITOR`.

```
~/loja$ export EDITOR
```

Agora, essa variável é uma variável de ambiente e se digitarmos `git commit` e dermos um "Enter" ele vai nos mostrar o `vi` .

Teremos o seguinte:

Se quisermos sair do `vi` , basta digitar `:q` e "Enter".

E se quisermos usar, novamente, o `nano` , que era o editor padrão DO `git` , digitamos `EDITOR=/bin/nano` e digitamos `git commit` e ele irá abrir o `nano` para nós.

Teremos o seguinte:

```
~/loja$ EDITOR=/bin/nano
~/loja$ git commit
```

E abrirá o `nano` :

Lembre-se que já exportamos a variável `EDITOR` . Ela está marcada para exportação, então, a variável editor funciona. Vamos sair do diretório `loja` . Para isso, utilizamos `cd ..` e retornamos ao nosso diretório inicial.

Repare que não estamos interessados em passar o funcionamento do `git` , não é o que queríamos passar, isso não é o mais importante, pois, temos um curso só sobre `git` . O importante aqui, é perceber que o `git` utiliza a variável de ambiente `EDITOR` , então, tínhamos que falar que `EDITOR` era esse. Toda vez que damos um `commit` puro e seco, ele usa para indicar qual é o `EDITOR` que utilizaremos para salvar um arquivo de acordo com a variável `EDITOR`.

Vimos a variável `HOME` , nosso diretório padrão, a variável `LOGNAME` que é o nosso usuário, a variável `UID` que é somente leitura e identifica o `id` do usuário e a variável `EDITOR` que indica um editor que podemos querer utilizar no dia a dia.

Ainda, veremos outras diversas variáveis e como é seu funcionamento no nosso dia a dia. Veremos isso aos poucos.